**一点都不SB的SBT**

惯例,先献上一篇[极品论文](http://www.docin.com/p-47348009.html)..

一种在编程复杂度,时间复杂度,空间复杂度都很优秀的平衡二叉树

这几天我是在学单调DP的时候 发现常常要对一个有序数列进行很多操作,一般的堆完成不了,于是找了二叉平衡树的资料学习.

本来庒神曾经给我splay tree的论文说很强大,但是经过比较发现SBT比伸展树各个方面都要强大…

算法和介绍都在上边论文里提到了.就不多说了

我就负责贴下自己写的模板还有一些练习的题目~

当然,还有很多题目,以下5题是较适合用平衡二叉树来解决的..其实最后一题就很鸡肋了,所以在很多没必要用SBT的时候还是别用好了,虽然SBT和同类算法比很优秀,但如果有其他方法能解决问题的时候用SBT就显得有点SB了..  
[[HNOI2002]营业额统计](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1588) {Insert , pred , succ , find}  
[[NOI2004]郁闷的出纳员](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1503) {Insert , DeleteSmall , Select}  
[[HNOI2004]宠物收养所](http://61.187.179.132:8080/JudgeOnline/showproblem?problem_id=1208) {Insert , pred , succ , Delete}  
[Double Queue](http://acm.pku.edu.cn/JudgeOnline/problem?id=3481) {Insert , DeleteMax , DeleteMin}  
[Buy Tickets](http://acm.pku.edu.cn/JudgeOnline/problem?id=2828) {DeleteSelect , build}

当然,这个模板在只能用于网络赛的时候贴一贴,现场赛的时候还是要理解了才能快速的敲出来~  
(一开始用指针的,但是体积是现在的两倍…现场赛时一定难敲了很多,遂改成数组模拟)

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[[Copy to clipboard]View Code](javascript:;) CPP

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232 | template<typename Type>  class SizeBalanceTree {  public:  void clear() {  sz = 0;  LC[0] = LC[0] = 0;  SZ[0] = 0;  root = 0;  }  int Size() {return SZ[root];}  bool empty() {return root == 0;}  void Build(int s,int e) {Build(root,s,e);}  bool Find(Type key) {return Find(root , key);}  void Insert(Type key) {Insert(root , key);}  void Delete(Type key) {Delete(root , key);}  Type DeleteSelect(int k) {return DeleteSelect(root , k);}  void DeleteSmall(Type key) {DeleteSmall(root , key);}  int Rank(Type key) {return Rank(root , key);}  Type Select(int k) {return Select(root , k);}  Type pred(Type key) {return pred(root , key);}  Type succ(Type key) {return succ(root , key);}  Type getMin() {  int temp = root;  while (LC[temp]) temp = LC[temp];  return K[temp];  }  Type getMax() {  int temp = root;  while (RC[temp]) temp = RC[temp];  return K[temp];  }  Type DeleteMax() {  int temp = root;  if(RC[root] == 0) {  root = LC[root];  return K[temp];  }  while (RC[RC[temp]]) {  SZ[temp] --;  temp = RC[temp];  }  SZ[temp] --;  Type ret = K[RC[temp]];  RC[temp] = LC[RC[temp]];  return ret;  }  Type DeleteMin() {  int temp = root;  if(LC[root] == 0) {  root = RC[root];  return K[temp];  }  while (LC[LC[temp]]) {  SZ[temp] --;  temp = LC[temp];  }  SZ[temp] --;  Type ret = K[LC[temp]];  LC[temp] = RC[LC[temp]];  return ret;  }  private:  int SZ[maxn];  Type K[maxn];  int LC[maxn];  int RC[maxn];  int root , sz;    void Build(int &root,int s,int e) {  if(s > e) return ;  int mid = (s + e)/2;  root = ++sz;  K[root] = mid;  LC[root] = 0;  RC[root] = 0;  SZ[root] = e - s + 1;  if(s == e) return ;  Build(LC[root] , s , mid - 1);  Build(RC[root] , mid + 1 , e);  }  bool Find(int &root,Type key) {  if (root == 0) {  return false;  } else if (key < K[root]) {  return Find(LC[root] , key);  } else {  return (K[root] == key || Find(RC[root] , key));  }  }  void Insert(int &root,Type key) {  if (root == 0) {  root = ++ sz;  LC[root] = RC[root] = 0;  SZ[root] = 1;  K[root] = key;  return ;  }  SZ[root] ++;  if (key < K[root]) {  Insert(LC[root] , key);  } else {  Insert(RC[root] , key);  }  maintain(root , !(key < K[root]));  }  Type Delete(int &root,Type key) {  SZ[root] --;  if ((K[root] == key) || (key < K[root] && LC[root] == 0) || (K[root] < key && RC[root] == 0)) {  Type ret = K[root];  if ( LC[root] == 0 || RC[root] == 0 ) {  root = LC[root] + RC[root];  } else {  K[root] = Delete(LC[root] , K[root] + 1);  }  return ret;  } else {  if ( key < K[root] ) {  return Delete(LC[root] , key);  } else {  return Delete(RC[root] , key);  }  }  }  void DeleteSmall(int &root , Type key) {  if ( root == 0 ) return ;  if ( K[root] < key ) {  root = RC[root];  DeleteSmall(root , key);  } else {  DeleteSmall(LC[root] , key);  SZ[root] = 1 + SZ[LC[root]] + SZ[RC[root]];  }  }  int Rank(int &root , Type key) {  if ( K[root] == key ) {  return 1;  } else if ( key < K[root] ) {  return Rank(LC[root], key);  } else {  return SZ[LC[root]] + 1 + Rank(RC[root] , key);  }  }  Type Select(int &root , int k) {  if ( SZ[LC[root]] + 1 == k ) {  return K[root];  } else if ( k > SZ[LC[root]] ) {  return Select(RC[root] , k - 1 - SZ[LC[root]]);  } else {  return Select(LC[root] , k);  }  }  Type DeleteSelect(int &root,int k) {  SZ[root] --;  if ( SZ[LC[root]] + 1 == k ) {  Type ret = K[root];  if (LC[root] == 0 || RC[root] == 0 ) {  root = LC[root] + RC[root];  } else {  K[root] = Delete(LC[root] , K[root] + 1);  }  return ret;  } else if ( k > SZ[LC[root]] ) {  return DeleteSelect(RC[root] , k - 1 - SZ[LC[root]]);  } else {  return DeleteSelect(LC[root] , k);  }  }  Type pred(int &root , Type key) {  if (root == 0) {  return key;  } else if (key > K[root]) {  Type ret = pred(RC[root] , key);  if(ret == key) return K[root];  return ret;  } else {  return pred(LC[root] , key);  }  }  Type succ(int &root , Type key) {  if (root == 0) {  return key;  } else if (K[root] > key) {  Type ret = succ(LC[root] , key);  if (ret == key) return K[root];  return ret;  } else {  return succ(RC[root] , key);  }  }  void LeftRotate(int &root) {  int temp = RC[root];  RC[root] = LC[temp];  LC[temp] = root;  SZ[temp] = SZ[root];  SZ[root] = 1 + SZ[LC[root]] + SZ[RC[root]];  root = temp;  }  void RightRotate(int &root) {  int temp = LC[root];  LC[root] = RC[temp];  RC[temp] = root;  SZ[temp] = SZ[root];  SZ[root] = 1 + SZ[LC[root]] + SZ[RC[root]];  root = temp;  }  void maintain(int &root , bool flag) {  if (root == 0) return ;  if ( !flag ) {  if ( SZ[LC[LC[root]]] > SZ[RC[root]] ) {  RightRotate( root );  } else if ( SZ[RC[LC[root]]] > SZ[RC[root]] ) {  LeftRotate( LC[root] );  RightRotate( root );  } else {  return ;  }  } else {  if ( SZ[RC[RC[root]]] > SZ[LC[root]] ) {  LeftRotate( root );  } else if ( SZ[LC[RC[root]]] > SZ[LC[root]] ) {  RightRotate( RC[root] );  LeftRotate( root );  } else {  return ;  }  }  maintain(LC[root] , false);  maintain(RC[root] , true);  maintain(root , false);  maintain(root , true);  }  }; |

如果需要用到int,double,longlong之外的一些类型的话,重载几个符号就好了,就像下边一样

[?](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[[Copy to clipboard]View Code](javascript:;) CPP

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | struct Node{  int key , priority;  Node() {}  Node(int a,int b):key(a) , priority(b) {}  bool operator < (Node b) {  return priority < b.priority;  }  bool operator > (Node b) {  return priority > b.priority;  }  bool operator == (Node b) {  return priority == b.priority;  }  Node operator + (int a) {  return Node(key , priority + a);  }  };  SizeBalanceTree<Node> SBT; |